Competitive Learning and Soft Competition for Vector Quantizer Design

Eyal Yair, Kenneth Zeger, and Allen Gersho, *Fellow, IEEE*

Abstract—**Vector quantizer (VQ) design is a multidimensional optimization problem in which a distortion function is minimized. The most widely used technique for designing vector quantizers is the generalized Lloyd algorithm (GLA), an iterative descent algorithm which monotonically decreases the distortion function towards a local minimum. One major drawback of the GLA, and of any descent minimization technique, is the "greedy" nature of the search, generally resulting in a nonglobal local optimum. A promising alternative to the GLA is the Kohonen learning algorithm (KLA), originally proposed for unsupervised training of neural networks. The KLA is an "on-line" algorithm where the codebook is designed while training data is arriving, and the reduction of the distortion function is not necessarily monotonic. In this paper we provide a convergence analysis for the KLA with respect to VQ optimality criteria and introduce a stochastic relaxation technique which produces the global minimum but is computationally expensive. By incorporating the principles of the stochastic approach into the KLA, a new deterministic VQ design algorithm, called the soft competition scheme (SCS), is introduced. Experimental results are presented where the SCS consistently provided better codebooks than the GLA, even when the same computation time was used for both algorithms. The SCS may therefore prove to be a valuable alternative to the GLA for VQ design.**

I. Introduction

VECTOR quantization (VQ) is a technique that can be used to map analog waveforms or discrete vector sources into a sequence of digital data for storage or transmission over a channel. A vector quantizer is a mapping of input vectors to one of a finite collection of predetermined codevectors, where the set of all codevectors is called a codebook. In designing a vector quantizer, the goal is a construct a codebook for which the expected distortion, introduced by approximating an input vector by a codevector, is minimized. The VQ design problem is a nonconvex optimization for which necessary conditions to attain optimality can be specified. For the mean-squared distortion function two such useful necessary conditions are the centroid and nearest neighbor conditions [1], [2].

Manuscript received September 8, 1989; revised July 31, 1990. This work was supported by the Weizmann Foundation for Scientific Research, the University of California MICRO program, Bell-Northern Research, Rockwell International, and Bell Communications Research.

E. Yair is with the IBM Scientific Center, Technion City, Haifa 32000, Israel.

K. Zeger is with the Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822.

A. Gersho is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106.

IEEE Log Number 9104891.

At present, the most widely used technique to design VQ codebooks is the generalized Lloyd algorithm (GLA) [3], a multidimensional generalization of the Lloyd scalar quantization algorithm [4]. The GLA is a descent algorithm for which a distortion function monotonically decreases by iteratively updating the codebook in an attempt to satisfy both the centroid and nearest neighbor rules. Unfortunately, since the distortion function is generally not convex and may contain multiple local minima [5], the GLA often produces nonoptimal codebooks. The existence of poor local minima is also demonstrated by the improvements over the GLA obtained in [6] using probabilistic VQ design methods (though with a significant increase in complexity).

A clustering technique, called the self-organizing feature map was proposed by Kohonen (see, for instance [7]) as a learning algorithm for neural networks. These self-organizing feature maps have also been used for VQ design [8]. Unlike the GLA, which is a batch algorithm, Kohonen learning is an on-line algorithm, which modifies the codebook each time a training vector is presented. As opposed to a batch algorithm in which the design of the codebook begins after the entire set of training data has arrived, in an on-line algorithm the codebook update occurs "on the fly," after each presentation of a training vector. Hereafter, Kohonen's learning scheme will be referred to as the Kohonen learning algorithm (KLA). The KLA is an application of the least mean squares (or LMS) algorithm [9], [10] to a nonquadratic cost function, which updates the codebook on-line using an instantaneous estimate of the gradient, known as the stochastic gradient. The KLA does not ensure that the distortion function decrease monotonically; it thus may avoid being trapped in poor local minima and if trained carefully may therefore find better solutions than a "greedy" algorithm, such as the GLA. Chang and Gray [11], [41] independently introduced an on-line technique for VQ design called the stochastic gradient (SG) algorithm, which is a special case of the KLA. Their experiments showed that the SG algorithm can yield slightly better codebooks than the GLA. Nasrabadi and Feng [8] used Kohonen learning to design vector quantizers for image coding, and similarly observed equal or slightly better results than with the GLA. Gersho [12] introduced an LMS-type algorithm for adapting a VQ codebook to changing source statistics.

To our knowledge, no analysis of Kohonen learning and its convergence properties with respect to VQ optimality

1053-587X/92$03.00

conditions has been published. Parameters, such as the reduction rate of the step size in the codevector update formula, are generally chosen by ad hoc techniques [7]. In the SG algorithm [11], the authors used a step-size formula suggested by Eweda and Macchi [13] for a linear adaptive filtering model. However, the filtering model is slightly different than the KLA (or the SG) and hence the convergence analysis given in [13] may not extend to the KLA. Moreover, although this step-size formula was reported to work well experimentally for the SG algorithm [11], there are no guidelines on how to choose some of the parameters. This led Chang and Gray to conclude that overall the GLA is preferable to use in practice.

In this paper, we first provide a convergence analysis for the KLA for a single codevector, and determine the conditions required to meet the centroid and nearest neighbor optimality criteria. The convergence for the more realistic case of multiple codevectors is very difficult to analyze and, in fact, although LMS-type algorithms are used extensively in many applications, there are no rigorous convergence proofs for nonquadratic distortion functions. However, assuming certain reasonable convergence properties of the KLA, the analysis given for a single codevector is valuable for assessing the asymptotic properties of the multiple codevector case.

We then introduce a stochastic relaxation scheme for VQ design, based on the Gibbs sampler of Geman and Geman [14] which, if given sufficient computation time, will produce a codebook with the global distortion minimum. The stochastic scheme, however, is generally very time consuming. By incorporating the principles of the stochastic scheme into the deterministic KLA, we then introduce a new on-line design algorithm, which can be viewed as a modification of Kohonen's neighborhood mechanism.[1] The resultant learning algorithm, called the soft competition scheme (SCS), is shown to be a valuable algorithm for designing vector quantizers, which consistently provide better codebooks than the GLA. Two additional significant advantages of the SCS are its suitability for parallel implementation and for real-time codebook adaptations (such as described in [12]).

The remainder of this paper is organized as follows. A presentation of the KLA and its use for VQ design is given in Section II. The asymptotic behavior of the KLA as an on-line estimator for a single codevector, and its implications for the multiple codevector case are discussed in Section III. Conditions on the step-size reduction rate in the adaptation formula are derived, and the unique schedule for which the Cramer–Rao lower bound is achieved for the estimator variance is obtained. The stochastic relaxation scheme is given in Section IV. The scheme is similar to the simulated annealing technique used by Zeger, Vaisey, and Gersho [6] but has a much higher transition rate and is amenable to parallel implementation. The soft competition scheme is introduced in Section V, where the principles of the stochastic scheme are incorporated into the deterministic KLA. This new algorithm is computationally equivalent in each update step to the original Kohonen learning but has the capability of finding higher quality solutions. Experimental results comparing the performance of the SCS to the GLA are given in Section VI for both Gauss–Markov and speech sources. These results demonstrate that the SCS can consistently provide better codebooks (in terms of mean-square error) than the GLA, even when the same amount of computation time is allocated for each of the algorithms.

[1]More details on the neighborhood mechanism are given in Section II.

II. Vector Quantization and Kohonen Learning

A. VQ Fundamentals and Notation

A vector quantizer Q is a mapping of the k-dimensional Euclidean space $\boldsymbol{R}^k$ into a finite subset $Y = \{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_K\}$ of $\boldsymbol{R}^k$. Each $\boldsymbol{y}_i$ is called a codevector and the set Y is called a codebook. The performance of a vector quantizer is measured by the average distortion, $D(Y) = E[d(\boldsymbol{x}, Q(\boldsymbol{x}))]$, between the source vectors and the quantized reproduction vectors, where $d(\cdot)$ is some distortion function, usually taken as the squared-error distortion $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|^2$, and $E[\cdot]$ denotes the expectation operator. The statistics of the source are assumed to be represented by a finite training set of vectors. The objective of the design procedure is to minimize the average distortion. Throughout the paper we assume that the mean-squared distortion function is used.

A vector quantizer is said to be optimal if no other quantizer has a smaller average distortion, for the same source, vector dimension, and codebook size. The following are two well-known necessary conditions for a vector quantizer to be optimal.

1) Nearest Neighbor Condition: For a given codebook, each training vector is mapped (quantized) to the codevector closest to it ("ties" are decided arbitrarily). This implies a partition of the space $\boldsymbol{R}^k$ into disjoint regions $(1 \le i \le K)$

$$R_i = \{\boldsymbol{x} \in \boldsymbol{R}^k \mid \|\boldsymbol{x} - \boldsymbol{y}_i\| < \|\boldsymbol{x} - \boldsymbol{y}_i\| \text{ for all } j \ne i\}. \tag{2.1}$$

Each partition region R_i is called the nearest neighbor cell of the ith codevector, and each $\boldsymbol{x} \in R_i$ is quantized to the codevector $\boldsymbol{y}_i$.

2) Centroid Condition: For a given partition $\{R_i\}$, each codevector $\boldsymbol{y}_i$ is the centroid of the cell R_i, given by $\boldsymbol{y}_i = E[\boldsymbol{x} | \boldsymbol{x} \in R_i]$. For a finite training set, the expected value is replaced by the sample mean of the ith cell.

If the nearest neighbor condition is assumed to hold, then the objective in designing a vector quantizer reduces to finding a codebook which minimizes the average distortion $D(Y)$. The GLA is a VQ design procedure which iteratively computes partition region centroids and performs nearest neighbor clustering to improve the quantizer performance for a given training set. The GLA is a batch-type algorithm; it takes into account all the training vectors in each iteration. As opposed to an on-line algorithm it cannot handle a continuous stream of training

data, or incorporate new source data without reprocessing all of the previous processed data. On-line algorithms can be used for real-time adaptation as source statistics change, without introducing large delays.

B. Kohonen Learning

A classical learning algorithm for self-organization of neural networks was proposed by Kohonen [7]. Kohonen's algorithm can also be used for designing vector quantizers [8]. Here we give a general formulation (and slight extension) of the learning scheme as applied to an on-line VQ design algorithm.

The codevectors $\boldsymbol{y}_i \in \boldsymbol{R}^k$ are associated with lattice nodes in a space with dimension smaller than k. The codebook is initialized (arbitrarily, except that no two codevectors can be equal) to $Y(0) = \{\boldsymbol{y}_1(0), \cdots, \boldsymbol{y}_K(0)\}$ and is iteratively updated as follows. At each iteration n, a training vector, $\boldsymbol{x}(n)$, is presented to the network, and the codevectors are modified according to the learning formula

$$\begin{aligned} \boldsymbol{y}_i(n) &= \boldsymbol{y}_i(n-1) + a_i(n)h_i(j^*, n)[\boldsymbol{x}(n) - \boldsymbol{y}_i(n-1)] \\ j^* &= \operatorname*{argmin}_j \|\boldsymbol{x}(n) - \boldsymbol{y}_j(n-1)\| \end{aligned} \tag{2.2}$$

where $a_i(n) \in [0, 1]$ is the step size of the ith codevector at time instant n, which is assumed to decay to zero as n increases. $\boldsymbol{y}_{j^*}$ is the codevector closest to $\boldsymbol{x}(n)$ in Euclidean space. The determination of $\boldsymbol{y}_{j^*}$ can be thought of as a competition between the codevectors to determine which is the nearest to $\boldsymbol{x}(n)$. Hereafter, $\boldsymbol{y}_{j^*}$ will be called the Euclidean winner for $\boldsymbol{x}(n)$. Other types of winners will be introduced in Section IV. Kohonen learning thus belongs to the class of competitive learning algorithms [15]–[18]. The function $h_i(j^*, n)$ is a *neighborhood* function that at time instant n is used to alter the step size of the ith codevector as a function of the physical distance of its associated node on the lattice from that of the winner j^*. Typically, $h_i(j, n)$ is nonzero for nodes close to j's on the lattice, and is zero outside this "neighborhood." Hence, only codevectors inside the neighborhood of the winner j^* are updated upon presentation of $\boldsymbol{x}(n)$. The size of neighborhood is reduced as a function of time in a predetermined manner. When the neighborhood is reduced to include only a single codevector which is the winner j^*, the update formula of (2.2) can then be written as

$$\boldsymbol{y}_i(n) = \boldsymbol{y}_i(n-1) + a_i(n)S_i(\boldsymbol{x}(n))[\boldsymbol{x}(n) - \boldsymbol{y}_i(n-1)] \tag{2.3a}$$

in which $S_i(\boldsymbol{x})$ are selection functions defined as

$$S_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in R_i \\ 0 & \text{else} \end{cases} \tag{2.3b}$$

and R_i is the nearest neighbor cell of the codevector $\boldsymbol{y}_i$ as defined in (2.1). In this case, upon presentation of a training vector $\boldsymbol{x}$, only the Euclidean winner is updated by moving towards that training vector with a step-size a_i. The SG algorithm presented in [11] used a similar update formula as in (2.3). In contrast to the SG algorithm or other applications of the Kohonen learning scheme (e.g., [19], [7], [8], [20]) in which the step-size $a(n)$ was independent of i, we allow each codevector to have its own step-size $a_i(n)$.

The neighborhood function is used to ensure that many of the codevectors are affected by each new training vector. If only the Euclidean winner is updated throughout the learning process, it might happen that some of the codevectors never win due to poor initial conditions. Such codevectors would then represent empty cells in the final codebook. Solutions to prevent such pathological situations, other than using a neighborhood mechanism, are given in [18], [19] in which the ability of each codevector to win is inversely related to its previous rate of winning. The neighborhood mechanism is used to achieve better partitions of the input space (or the training set) and, thus, better local minima of the distortion function. In Section IV and V we develop stochastic and deterministic types of neighborhoods that can find the global minimum of $D(Y)$ for a given training set. However, for analyzing the convergence of $D(Y)$ to a local minimum (i.e., for satisfying the centroid and nearest neighbor conditions) it suffices to assume that only the Euclidean winner is updated in each step. To this end we proceed with the analysis of (2.3).

Let n_i denote the number of times that the ith codevector $\boldsymbol{y}_i$ has won and been updated. That is, each codevector has an independent time counter which is incremented only after that codevector is updated. Formally, the update of these counters at the nth step can be written as $(1 \le i \le K)$

$$n_i(n) = n_i(n-1) + S_i(\boldsymbol{x}(n)). \tag{2.4}$$

Hereafter, $\boldsymbol{y}_i(n)$ will denote the value of the ith codevector after n updates $(n = 1, 2, \cdots)$ where it is understood that n represents the value of the counter n_i. With this notation, the update formula of (2.3) for the ith codevector can be written as

$$\boldsymbol{y}_i(n) = \boldsymbol{y}_i(n-1) + a_i(n)[\boldsymbol{x}_n^{(i)} - \boldsymbol{y}_i(n-1)] \tag{2.5}$$

where $a_i(n)$ is the value of the step size for the ith codevector for $n_i = n$, and $\boldsymbol{x}_n^{(i)}$ is the nth training pattern for which the ith codevector wins. Thus, unlike the original Kohonen learning for which $a_i(n)$ is independent of i, each codevector, at any instant of the algorithm in (2.5), will have a different value for the step size. Equation (2.5) can be also written as

$$\boldsymbol{y}_i(n) = [1 - a_i(n)]\boldsymbol{y}_i(n-1) + a_i(n)\boldsymbol{x}_n^{(i)} \tag{2.6}$$

whose solution after N updates is given by

$$\boldsymbol{y}_i(N) = A_i(0, N)\boldsymbol{y}_i(0) + \sum_{n=1}^{N} A_i(n, N)a_i(n)\boldsymbol{x}_n^{(i)} \tag{2.7a}$$

where for $0 \le n \le N-1$

$$A_i(n, N) = \sum_{j=n+1}^{N} [1 - a_i(j)] \quad \text{and } A_i(N, N) = 1. \tag{2.7b}$$

A key problem with this algorithm is how to reduce the step-size values $a_i(n)$ to zero so that a codebook is produced which satisfies both the nearest-neighbor and the centroid conditions. We call the functional decay of $a_i(n)$ the step-size schedule of the algorithm. In Section III we address the question of what step-size schedule should be used for the KLA.

III. Asymptotic Properties

In the following discussion we assume that the training vectors are generated by a stationary memoryless source with an unknown probability distribution. That is, the training patterns are independent, identically distributed (i.i.d.) random vectors, denoted by $\boldsymbol{x}(n)$. We will examine the asymptotic behavior of the KLA as $n \to \infty$. Unfortunately, analysis of the general case is very difficult. Therefore, we first assume that there is only one codevector in the codebook. Later we show how the analysis of a single codevector can be used to address the more realistic problem of multiple codevectors.

After N updates, the value of the codevector $\boldsymbol{y}(N)$ is given, according to (2.7), by

$$\boldsymbol{y}(N) = A(0, N)\boldsymbol{y}(0) + \sum_{n=1}^{N} q_N(n)\boldsymbol{x}(n) \tag{3.1}$$

where

$$q_N(n) = A(n, N)a(n) \tag{3.2}$$

and $A(n, N)$ is given in (2.7b) (the index i is omitted). To satisfy the centroid condition, the value of the codevector should equal the expected value of $\boldsymbol{x}$. Hence, for any N, $\boldsymbol{y}(N)$ is an on-line estimate of the mean of $\boldsymbol{x}$ based on N measurements. The expected accuracy of the estimation at time N may be assessed by two important factors: the bias $b(N)$ of the estimator, and its variance, $\sigma_y^2(N)$. Since the training vectors are assumed i.i.d., these two quantities are given by

$$b(N) \triangleq E[\boldsymbol{y}(N)] - E[\boldsymbol{x}] = A(0, N)\boldsymbol{y}(0) + E[\boldsymbol{x}](r(N) - 1) \tag{3.3a}$$

where

$$r(N) = \sum_{n=1}^{N} q_N(n) \tag{3.3b}$$

$$\sigma_y^2(N) \triangleq \text{VAR}\,(\boldsymbol{y}(N)) = u(N)\,\text{VAR}\,(\boldsymbol{x}) \tag{3.4a}$$

$$u(N) = \sum_{n=1}^{N} q_N^2(n). \tag{3.4b}$$

To obtain a good estimator in the MSE (mean-squared error) sense, it is required that the sequences $b(N)$ and $\sigma_y^2(N)$ converge to zero as $N \to \infty$. The question discussed in this section is the following: what conditions on the step-size schedule $a(n)$ will guarantee that both the bias and variance of the estimator $\boldsymbol{y}(N)$ will converge to zero as $N \to \infty$? We assume hereafter that $a(n)$ is monotonically nonincreasing, decaying to zero as $n \to \infty$.

From (3.3) and (3.4) it is evident that the following three requirements must be satisfied:

$$\text{i)} \lim_{n \to \infty} A(0, n) = 0 \tag{3.5a}$$

$$\text{ii)} \lim_{n \to \infty} r(n) = 1 \tag{3.5b}$$

$$\text{iii)} \lim_{n \to \infty} u(n) = 0. \tag{3.5c}$$

Conditions i and ii make the bias zero and condition iii makes the variance zero. The next five theorems provide conditions for the step-size schedule to comply with the above three requirements. Formal proofs of these theorems are given in Appendix A.

Theorem 1: $\boldsymbol{y}(N)$ in (3.1) is asymptotically unbiased if and only if

$$\sum_{n=1}^{\infty} a(n) = \infty \tag{3.6}$$

and is unbiased for any value of N if $a(1) = 1$.

To achieve an unbiased estimator it is thus sufficient to choose a step-size schedule for which $a(1) = 1$. However, (3.6) must be satisfied since it is also a necessary condition for the convergence of the estimator variance to zero, as stated in Theorem 2:

Theorem 2: If $u(n)$ converges to zero then $\Sigma_{n=1}^{\infty} a(n) = \infty$.

In Appendix A-2 (Lemma 3) it is shown that if $a(n)$ decreases monotonically, then the variance sequence $u(n)$ also decreases monotonically. Thus, for any monotonic step-size schedule, an improvement in the estimation will be obtained as more observations become available. However, if the step-size schedule decreases too quickly the variance will converge to a nonzero value. Theorem 3 provides a sufficient condition for the convergence of the estimator variance to zero.

Theorem 3: A sufficient condition for the convergence of $u(n)$ to zero is

$$a(n + 1) \geq \frac{a(n)}{1 + a(n)} \quad \text{for all } n \geq 1. \tag{3.7}$$

While the class of step-size schedules for which the estimator variance converges to zero might be fairly large (e.g., all schedules that satisfy (3.7)), it is desirable to find the schedule with the lowest convergence time. We first give a bound on the convergence rate of the variance sequence of an unbiased estimator.

Theorem 4: If $r(N) = 1$ for all $N \geq 1$, then

$$u(N) \geq \frac{1}{N} \quad \text{for all } N \geq 1 \tag{3.8}$$

and equality is obtained if and only if $q_N(n) = 1/N$ for all $n \in [1, N]$.

This lower bound on the rate of decay of the variance is in fact the Cramer–Rao bound for unbiased estimators [21]. Note that an unbiased estimator is obtained simply by setting $a(1) = 1$. Hence, the most efficient estimator is obtained for a schedule $a(n)$ for which $q_N(n)$ is identically $1/N$ for all $n \in [1, N]$. In fact, the unique schedule that meets this condition is the schedule $a(n) = 1/n$, as given by Theorem 5.

Theorem 5: The step-size schedule $a(n) = 1/n$ is the unique schedule for which the Cramer–Rao bound is achieved for the variance of an unbiased estimator.

In the following subsections, three classes of schedules that are often used in practice are examined and their asymptotic behavior is analyzed (with respect to the above results).

A. Hyperbolic Schedule

Here we consider a hyperbolic schedule $a(n) = n^{-r}$ with $r > 0$.

Theorem 6: For a hyperbolic schedule, a necessary and sufficient condition to satisfy all three requirements of (3.5) is $r \leq 1$.

It is interesting to examine the asymptotic behavior of the weighting function $q_N(n)$ for a hyperbolic schedule. In a similar manner as in the proof of Lemma 4 (given in Appendix A-6), it can be shown that for $r \leq 1$ and $n \geq 2$,

$$\frac{q_N n}{q_N(n-1)} = \frac{(n-1)^r}{n^r - 1} \geq 1 \tag{3.9}$$

and equality holds only for $r = 1$. For $r = 1$, the centroid is estimated by the sample mean of all the data presented, while for $r < 1$, (3.9) implies that the weighting sequence $q_N(n)$ is an increasing sequence, with the learning algorithm thus giving more weight to the latest data presented.

B. Exponential Schedule

Here we consider an exponential schedule $a(n) = \lambda\rho^n$, with $\rho < 1$, and $\lambda \in (0, 1]$. For this schedule, the sequence $A(0, n)$ doe not converge to zero as $n \to \infty$ since (3.6) is violated. However, in practice, $A(0, n)$ can be made arbitrarily small by choosing ρ sufficiently close to unity: the inequality of (A.1) (given in Appendix A) can be used to give an upper bound on $A(0, n)$ as

$$A(0, n) = \prod_{i=0}^{n} (1 - \lambda\rho^i) < \exp\left\{-\lambda \sum_{i=0}^{n} \rho^i\right\} \underset{n\to\infty}{\to} \exp\left\{\frac{-\lambda}{1-\rho}\right\}. \tag{3.10}$$

This upper bound is finite for any $\rho < 1$, and can be made arbitrarily small by choosing ρ sufficiently close to unity. For example, to upper bound $A(0, n)$ by 10^{-6} (for $\lambda = 1$), ρ should be at least 0.93. By using the Taylor series expansion of $\ln(1 - x)$, the exact expression for the limit of $A(0, n)$ can be written as

$$\lim_{n\to\infty} A(0, n) = \exp\left\{-\sum_{i=1}^{\infty} \frac{1}{i} \frac{\lambda^i}{1-\rho^i}\right\} \tag{3.11}$$

and any finite sum in the above exponent will serve as a tighter upper bound than the one in (3.10). For example, by also considering the quadratic term in the Taylor series, the bound of 10^{-6} is achieved for $\rho = 0.91$. Thus in practice, for $\rho > 0.9$, the estimate can be considered to be asymptotically unbiased. However, while the asymptotic value of $A(0, n)$ is small, the learning algorithm weights the training vectors presented to the system in an undesirable way. Specifically, for large values of n, the most recent data presented is weighted less than previous data. Informally, this means that for some value of n the learning algorithm "freezes," and the presentation of new data will have a diminishing effect on the codevector locations. To show this we note that from (3.2) and (2.7b) it follows that for any k

$$\lim_{n\to\infty} \frac{q_N(n)}{q_N(n-k)} = \rho^k < 1 \tag{3.12}$$

which is in contrast to the hyperbolic case in (3.9). For large values of n we obtain $q_N(n) < q_N(n-1)$ since $q_N(n) \approx q_N(n-1)$, and therefore the most recent data presented is weighted less than older data. There exists a critical value of n (independent of N), denoted by n_c, after which newer inputs are weighted less than older ones. We call n_c the freezing instant, since thereafter new data presented to the network is exponentially attenuated and the algorithm progressively stops learning. The freezing instant can be explicitly obtained as the solution of $\rho = 1 - \lambda\rho^n$ (for which $q_N(n) = q_N(n-1)$), yielding

$$n_c = \frac{\ln(1-\rho) - \ln\lambda}{\ln\rho}. \tag{3.13}$$

For example, for $\rho = 0.9$ and $\lambda = 1$, we have $n_c = 22$, or for $\rho = 0.99$ we get $n_c = 458$. The freezing instant, n_c, becomes even lower for $\lambda < 1$, since n_c decreases with decreasing λ. For any $\lambda \leq 1 - \rho$, the freezing instant becomes zero, inducing the algorithm to start freezing at the very first input data. The existence of a freezing point independent of N, beyond which the incoming data is weighted less and less, implies that an exponential schedule is inadequate for VQ design (and for other self-organization applications using the KLA).

It is also interesting to inspect the freezing instant considering condition (3.7). For Theorem 3 it is evident that the variance sequence $u(n)$ is bounded by $a(n)$ when

$$\lambda\rho^{n+1} \geq \frac{\lambda\rho^n}{1 + \lambda\rho^n}. \tag{3.14}$$

By using the definition of the freezing instant in (3.13), the inequality in (3.14) can be expressed as

$$n \leq n_c - 1. \tag{3.15}$$

Thus, for the exponential schedule, freezing begins when condition (3.7) is violated.

C. Linear Schedule

Finally, we consider a linear decay of $a(n)$ of the form $a(n) = \lambda(1 - n/N)$ for $0 \leq n \leq N$ and $\lambda \in (0, 1]$, where the algorithm terminates after N steps. Asymptotically, as N becomes larger, condition (3.6) holds and the estimate becomes unbiased for any $\lambda \in (0, 1]$.

To examine the behavior of the weighting function $q_N(n)$ we note that

$$\frac{q_N(n)}{q_N(n-1)} = \frac{N-n}{N-n+1} \cdot \frac{N}{N-\lambda(N-n)}. \quad (3.16)$$

For small values of n ($n \ll N$) and for $\lambda = 1$, the above ratio can be approximated by $N/n \gg 1$, and for $\lambda < 1$ this ratio is approximately $(1 - \lambda)^{-1} > 1$. Hence, at the beginning of learning, the step-size schedule is too slow, and $q_N(n)$ increases faster as λ approaches unity. On the other hand, when $n \approx N$, the ratio in (3.16) is approximated by $(N - n)/(N - n + 1) < 1$. Using the boundary condition for $q_N(n - 1)$, the weighting factor $q_N(n)$ can be approximated for values of n close to N by $q_N(n) = 1 - n/N$, which decays linearly to zero. Therefore, asymptotically as n approaches N, the decay is too fast. Since the decay of the step size is too slow at the beginning of the learning and too fast at the end, there is some intermediate instant, for which the weighting sequence $q_N(n)$ peaks. In other words, in the linear schedule, the weighting of the data is biased in favor of data presented around some intermediate instant, denoted, by n_r, for which a maximum in the sequence $q_N(n)$ is attained. To solve for n_r, the ratio in (3.16) is unity and we obtain

$$n_r = N - \sqrt{\frac{N}{\lambda}}. \quad (3.17)$$

Note that n_r decreases as λ decreases, so the weighting function $q_N(n)$ peaks for "older" input data. Unlike the exponential case in which n_c was a fixed point, independent of N, the critical point n_r is relatively close to N. Hence, while a learning algorithm with the exponential schedule tends to favor data presented at the beginning of the learning and ignores the most recent data, a linear schedule favors the recent data (see Fig. 1).

Finally, by the checking condition (3.7), it can be verified that as in the exponential case, $u(n)$ is bounded by $a(n)$ for $n < n_r$, and the condition is violated for $n > n_r$. A graphical illustration of the behavior of $q_N(n)$, demonstrating the typical shape of the weighting sequence as a function of n, is given in Fig. 1 for the exponential, hyperbolic, and linear step-size schedules.

D. The Nonquadratic Case

In the analysis in Sections III-A through III-C we assumed there was only one codevector, which implied that the distortion surface was quadratic and therefore convex.

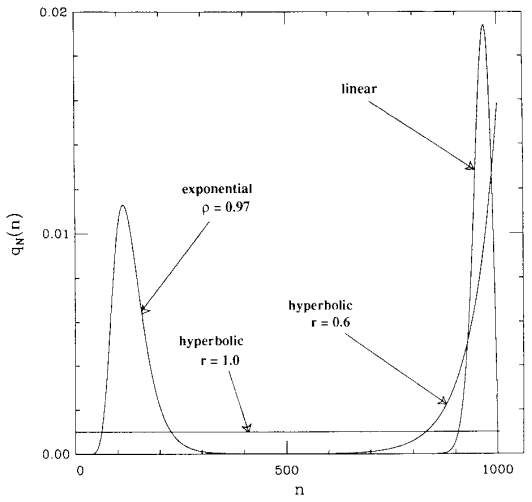


Fig. 1. Illustration of the behavior of the weighting sequence $q_N(n)$ as a function of n(with $N = 1000$) for four step-size schedules: 1) Hyperbolic schedule with $r = 1$ for which a constant weighting of the training data is obtained throughout the whole learning process. This is the optimal weighting. 2) Hyperbolic schedule with $r = 0.6$ which favors more recent data but still, asymptotically, achieves the centroid of the training vectors. 3) Exponential schedule with $\rho = 0.97$ for which there is an early termination of learning due to the "freezing" effect, and the weighting of the most recent data is practically zero. 4) Linear schedule which, as opposed to the exponential schedule, effectively weights only the most recent data.

With multiple codevectors, the distortion surface is generally no longer convex and contains multiple local minima. Assuming that the algorithm converges, as time progresses the step size becomes sufficiently small and the nearest neighbor partition of the input space induced by the codebook essentially does not change. The asymptotic behavior of each codevector in its own cell then obeys the asymptotic properties discussed for a single codevector. Convergence of each codevector to the centroid of its nearest neighbor cell implies that the centroid condition is satisfied and the codebook then represents a local minimum of the distortion function.

While we have informally argued in favor of the convergence of the KLA for the nonquadratic case, a rigorous proof of its convergence is difficult to construct since it is hard to measure the change in the partition cells when the codevectors are updated. Unlike true gradient algorithms, the reduction of the distortion function in the KLA is not necessarily monotonic (even asymptotically). The KLA of (2.5), being the on-line version of a batch gradient descent algorithm, uses the instantaneous gradient, not the true gradient. In fact, the KLA is an application of the LMS learning principle to a nonquadratic distortion function. Such on-line LMS algorithms are also known as stochastic-gradient algorithms, for which no mathematical proof of convergence is yet known for the nonquadratic case. However, in practice, such algorithms have been successfully used in a variety of applications for nonquadratic cost functions, in which convergence of the learning scheme was always obtained, once reasonable values for parameters were determined. Examples involv-

ing Kohonen learning can be found in speech applications [22, 23], robotics and pattern recognition [20], and VQ design [11, 18]. Applications with another on-line LMS-type learning, the backpropagation algorithm [24], are found in speech applications [25]–[28], character recognition [29], [30], and sonar applications [31]. It was recently reported that such an on-line adaptation for the backpropagation algorithm was even found to outperform the batch adaptation in applications of handwritten character recognition [30], [32].

In the above examples, the least mean-squares criterion was used as the cost function. Recently, a proof of convergence of an on-line learning algorithm for a Kullback–Leibler type of cost function (defined in [33]) which is used in Boltzmann machines was given by Sussmann [34]. Based on this empirical experience it is reasonable to assume that the KLA converges, and thus, asymptotically each codevector wins only for training vectors in its own cell. The asymptotic convergence of each codevector to the cell's centroid is therefore subject to the convergence analysis for a single codevector discussed above. Different step-size schedules might lead to different final codebooks (as noticed in [11]) but these different codebooks will represent different local minima of $D(Y)$ if the schedules comply with the requirements for convergence given above. A local minimum of the distortion function does not necessarily guarantee a high quality codebook, as it may differ substantially from a global minimum. In the remainder of the paper, the problem of finding local minima of good quality is addressed by incorporating soft competition techniques for which the competition between the codevectors upon each presentation of a training vector is "soft," and the codevectors being updated are not necessarily the Euclidean winners.

IV. The Stochastic Relaxation Scheme

So far we have described VQ design as a procedure that locates K codevectors in an input space that locally minimizes a mean-squared distortion function $D(Y)$ given by

$$D(Y) = \sum_{i=1}^{K} \sum_{X \in R_i} \|\boldsymbol{x} - \boldsymbol{y}_i\|^2 \tag{4.1}$$

while for each codebook the nearest neighbor condition was assumed. An alternative approach is to search for the best partition, while assuming the centroid condition is satisfied. Let us assume that the training set $\boldsymbol{T}$ is composed of M training vectors, $\boldsymbol{T} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M\}$. A partition of the training set into K distinct regions is accomplished by associating with each training vector $\boldsymbol{x}_m$ a scalar $s_m \in \{1, \cdots, K\}$, which indicates the region index of $\boldsymbol{x}_m$. Hence, the partition of the training set can be fully described by the M-dimensional state vector $\boldsymbol{s} = (s_1, \cdots, s_M)$ and the K partition regions R_i are given by ($1 \le i \le K$)

$$R_i = \{\boldsymbol{x}_m \in \boldsymbol{T} | s_m = i\} \tag{4.2}$$

which are not necessarily the nearest neighbor cells of the codevectors. For a given state vector $\boldsymbol{s}$, the codevectors are assumed to be the centroids of the partition regions:

$$\boldsymbol{y}_i = \frac{1}{N_i} \sum_{\substack{m \\ s_m = i}} \boldsymbol{x}_m; \qquad N_i = \sum_{\substack{m \\ s_m = i}} 1. \tag{4.3}$$

By assuming that (4.3) holds for every partition vector $\boldsymbol{s}$, the codebook can be described uniquely by $\boldsymbol{s}$, and a distortion value $D(\boldsymbol{s})$ can be associated with each state $\boldsymbol{s}$, which computes the same expression as in (4.1) via (4.2) and (4.3). The goal of the design in this case is to find a choice of $\boldsymbol{s}$ which globally minimizes $D(\boldsymbol{s})$.

A general way to find the global minimum of a multidimensional nonconvex cost function $D(\boldsymbol{s})$ defined over a finite state space was introduced by Kirkpatrick *et al.* [35] under the name of simulated annealing. The simulated annealing scheme is based on the Metropolis algorithm [36] suggested for simulating the dynamics of a melted substance in a heat bath. The algorithm is a stochastic procedure aimed at reducing the cost function in a nonmonotonic way, with the expressed goal to avoid local minima. At each instant of the algorithm, a new state is proposed, generally by changing the value of one of the components of the state vector $\boldsymbol{s}$, and the resulting change in the cost function ΔD is computed. The acceptance of the proposed transition is determined probabilistically as follows: if $\Delta D < 0$ the proposed state is accepted and the cost function is thereby reduced, and if $\Delta D > 0$ the proposed state is accepted with probability $p = \exp(-\Delta D/T)$, and rejected with probability $1 - p$. The parameter T is a nonnegative number called the temperature and is gradually reduced to zero. As T decreases, the probability of making transitions that increase the cost function also decreases, and in the limit as $T \to 0$ the cost function can only be decreased. The rate at which T is reduced is called the temperature schedule of the annealing.

The above procedure represents an inhomogeneous finite-state Markov chain which, if cooled sufficiently slowly, converges to the global minimum of $D(\boldsymbol{s})$ [37]. A key factor to ensure the convergence of such a Markov chain to the global minimum of $D(\boldsymbol{s})$ is that the stationary joint probability distributed of the states be the Gibbs distribution, given by

$$P(\boldsymbol{s}) = \frac{1}{Z} e^{-\beta D(\boldsymbol{s})} \tag{4.4}$$

where $P(\boldsymbol{s})$ is the probability of attaining a specific state $\boldsymbol{s}$, $\beta = 1/T$, and Z is a normalization factor defined so that the sum of $P(\boldsymbol{s})$ over all possible states in unity. More generally, any stochastic transition rule between the states which yields a Markov chain with the Gibbs stationary distribution of (4.4) will converge to the global minimum of $D(\boldsymbol{s})$, if T is reduced sufficiently slowly.

An example of a transition rule different than the Metropolis algorithm is the "Gibbs sampler" introduced by

Geman and Geman [14] for the restoration of images. The Gibbs sampler generalizes the binary Boltzmann machine developed by Hinton *et al.*, [38], [39] to nonbinary cases. Such algorithms are also known as stochastic relaxation schemes. The Gibbs sampler is better suited to problems in which the components of the state s are not binary and can take on any of K values. The case we study falls into this category, and we will show that by applying the Gibbs sampler, a stochastic scheme is obtained in which each of the codevectors is updated probabilistically. As opposed to the winner-take-all competition of previous sections, in which only the Euclidean winner was updated, at each iteration any codevector can be updated in the stochastic scheme. The probability of updating a codevector during a given iteration is a function of its distance from the training vector presented at that instant. We call this scheme the stochastic relaxation scheme (SRS), which will converge to the global minimum of $D(s)$ for certain temperature schedules. The fact that the codevectors being updated are not necessarily the Euclidean winners is the key to the capability of the algorithm to avoid local distortion minima.

An important advantage of using the Gibbs sampler for the nonbinary case, rather than the Metropolis algorithm, is that the Gibbs sampler is amenable to parallel processing. The Metropolis algorithm is serial in nature whereas the Gibbs sampler can incorporate K processors operating in parallel for the K possible values of s_m, and thereby speed up the computation.

The Gibbs sampler algorithm sequentially generates new state vectors by changing one component of s at a time. At each iteration a component $m \in \{1, \cdots, M\}$ of s is chosen at random. The new value of s_m is then set equal to some $j \in \{1, , \cdots, K\}$ drawn probabilistically according to the following conditional distribution ($1 \leq j \leq K$):

$$P_m(j) \triangleq \Pr\{s_m = j | s_n \quad \forall\, n \neq m\} \tag{4.5}$$

where the values of s_n (for all $n \neq m$) are those of the current state vector (which are given constants at the time instant). The conditional distribution $P_m(j)$ should be computed to comply with the Gibbs distribution given in (4.4). In Appendix B, it is proven that $P_m(j)$ can be expressed in terms of the codebook as ($1 \leq j \leq K$)

$$P_m(j) = \frac{e^{-\beta\|x_m - y_j\|^2}}{\sum_{k=1}^{K} e^{-\beta\|x_m - y_k\|^2}}. \tag{4.6}$$

Note that the conditional distribution, according to which the next partition of the training vector x_m will be chosen, as in (4.5), is also Gibbs, where the denominator in (4.6) is a normalization factor, independent of j. When $\beta \to \infty$ (i.e., the temperature goes to zero), this probability distribution becomes a delta function around the Euclidean winner (i.e., the codevector closest to x_m). In this case, the distortion function is strictly decreasing and the partition becomes the nearest neighbor partition, in which each training vector is assigned to the codevector closest to it. The expression in (4.6) is identical to the soft competition scheme introduced by Yair and Gersho for the Boltzmann perceptron network [40]. That is, $P_m(j)$ represents a fraction of unity which is proportional to the proximity of y_i to the current data presented x_m. It was also shown in [40] that this scheme can be efficiently implemented in parallel by a two-layer feedforward neural network.

Given below is an outline of the main loop of the SRS carried out repeatedly until the temperature reduces to zero.

1) Choose, at random, a training vector $x_m \in T$. Assume that the current partition of this vector is given by $s_m = i$.

2) Compute the K values $P_m(j)$ according to (4.6), and draw an index $j^* \in \{1, \cdots, K\}$ probabilistically according to this distribution.

3) Set $s_m = j^*$, and update the codevectors y_i and y_{j^*} according to the following formula (derived in Appendix B):

$$y_i \leftarrow y_i - \frac{1}{N_i}(x_m - y_i) \tag{4.7a}$$

$$y_{j^*} \leftarrow y_{j^*} + \frac{1}{N_j + 1}(x_m - y_{j^*}). \tag{4.7b}$$

4) Update β according to the temperature schedule.

The codevector y_{j^*} is called the stochastic winner (which is not necessarily the Euclidean winner), and y_i is called the stochastic loser. For each presentation of a training vector the stochastic winner thus moves towards x_m (as in the usual KLA), while the stochastic loser moves away from x_m.

The computational advantage of this algorithm over the Metropolis one should be noted. The efficiency of stochastic algorithms of these type can be expressed by the transition rate, which is the rate at which the network makes transitions to new states. The lower the transition rate, the less efficient is the algorithm in searching for the optimal solution. While the Metropolis algorithm must be performed serially and has a low transition rate (especially when K is large), the above stochastic relaxation scheme has a much higher transition rate (since transitions are performed unconditionally), and the soft competition of (4.6) can be computed in parallel by a neural network. When the temperature is low, most of the proposed states will not be accepted in the Metropolis algorithm and much computation will be (in effect) wasted staying in the same state. This situation will not happen in the SRS since s_m is set unconditionally to its new value.

V. The Soft-Competition Scheme

Inspired by the SRS algorithm presented in Section IV, in this section we develop a deterministic algorithm that updates all the codevectors simultaneously, rather than one at a time. We first relax the requirement that the

codevectors be restricted to the centroids of the current partition at any given instant. We also note that the current value of the state component s_m has no effect on its next value. Upon winning the stochastic competition of (4.6), y_{j^*} is updated with a KLA-type formula towards the input vector. The probability, however, of updating a codevector y_j is independent of the current affiliation of the data x_m. Hence, we can simplify the computation by not tracking the partition of the training vectors. Thus, upon each presentation of a training vector only the stochastic winner will be updated towards the training vector. However, instead of updating at each iteration only one codevector, chosen stochastically from a local Gibbs distribution, and moving it with a step-size $a(n)$ towards the data, an alternative (deterministic) approach is taken. At each step, all the codevectors are simultaneously updated towards the data, but for each of them the step size is scaled by the probability of winning, as given in (4.6). This algorithm, which we refer to as the soft competition scheme (SCS), can be formulated as follows.

The training vectors are presented on-line. At any time instant n, for which a training vector $x(n)$ is presented, the codebook is updated using $(1 \leq i \leq K)$

$$y_i(n) = y_i(n-1) + a_i(n)P_n(i)[x(n) - y_i(n-1)] \tag{5.1}$$

in which $P_n(i)$ is given by

$$P_n(i) = \frac{e^{-\beta(n)\|x(n) - y_i(n-1)\|^2}}{\sum_{k=1}^{K} e^{-\beta(n)\|x(n) - y_k(n-1)\|^2}} \tag{5.2}$$

and

$$\lim_{n \to \infty} \beta(n) = \infty. \tag{5.3}$$

That is, upon each presentation of a training vector $x(n)$, all the codevectors are simultaneously updated such that ith codevector is shifted a fraction of $P_n(i)$ from its prescribed step-size $a_i(n)$ at each step. Since all the codevectors are now updated at each step, all the counters of the K codevectors are also updated at time n. The counter update formula is given by $(1 \leq i \leq K)$

$$n_i(n) = n_i(n-1) + P_n(i) \tag{5.4}$$

which is the natural generalization of (2.4), where $n_i(n)$ is the value of the counter of the ith codevector at time instant n, and $a_i(n)$ is computed, as for the KLA, by $a_i(n) = 1/n_i(n)$.

This scheme is a "soft" competition scheme in the sense that there is no ultimate "winner," but rather, each codevector is updated towards the data with a step size that is proportional to its probability of winning. The algorithm starts with a low value of β, for which $P_n(i)$ is approximately uniform. That is, for low values of β (high temperatures) the codevectors are not yet attracted to a certain partition, and they all migrate towards the data presented. As time progresses, the gain β is gradually increased and the codevectors are slowly separated from each other since $P_n(i)$ begins peaking around the Euclidean winner. In time, the codevectors closer to the data will take relatively larger steps, while the codevectors which are far away will be increasingly less affected. In the limit as $n \to \infty$, $P_n(i) \to S_i(x(n))$, the competition becomes a winner-take-all competition and (5.1) is reduced to (2.3), in which only the Euclidean winner is updated. Comparing (5.1) to (2.2), the SCS is seen to be a neighborhood learning scheme in which $P_n(i)$ is the neighborhood function, aimed at improving the partition of the training set. However, unlike the neighborhood function $h_i(j, n)$ defined by Kohonen on a fixed lattice, in the SCS, $P_n(i)$ is a dynamic neighborhood function which weighs the codevectors by their distance from the data in the input space, rather than by their distance on the lattice. Since this algorithm is the deterministic equivalent of the Gibbs sampler introduced in the previous section, it can avoid local minima because the attraction of the codevectors to a specific partition evolves gradually, allowing them to find better partitions as time progresses. It should be noted that the computational complexity of the soft competition algorithm of (5.1) is essentially the same as the KLA of (2.3), as both involve the computation of K Euclidean distances for each presentation of a training vector. It can be conjectured that, as in simulated annealing, the schedule for the gain $\beta(n) = C \ln (n)$ achieves the global minimum of $D(s)$, although no proof is known to the authors.

Since the SCS asymptotically evolves into the KLA, the step-size schedule $a_i(n)$ should asymptotically be $1/n_i$ to ensure the centroid and nearest neighbor conditions for the final code. Such asymptotic behavior can be accomplished as follows. The time axis is segmented into frames which, for convenience, might be measured by the number of sweeps they contain. Each sweep is one full cycle through the training set. The lengths of the frames increase as the temperature $T(n) = 1/\beta(n)$ decreases, and the value of the step-size $a_i(n)$ in each frame is reduced according to $1/n_i$. The counters n_i are reinitialized to unity at the beginning of each frame and are incremented according to (5.4) within the frame. The reinitialization technique was found to be useful in practice to speed up the convergence of the algorithm. The increase of the frames' lengths ensures that more updates will be performed at low temperatures before the next counter initialization. Asymptotically, this technique ensures that the final codebook will satisfy both the centroid and the nearest neighbor conditions since (5.1) then becomes (2.3) as $T(n) \to 0$.

Finally, we give below a schematic outline of the algorithm steps carried out for each presentation of a training vector $x(n)$.

1) Compute $P_n(i)$ for $i = 1, \cdots, K$ using (5.2).
2) Update the counters n_i according to (5.4), and set the step-size values to $a_i(n) = 1/n_i(n)$.

3) Update the codevectors $\mathbf{y}_i$ for $i = 1, \cdots, K$ using (5.1).
4) Update $\beta(n)$ according to the temperature schedule.

Generally, the temperature update will be performed once for each sweep, and a reinitialization of the counters n_i is carried out at the end of each frame. Experimental results showing an increase in performance of the SCS over the GLA algorithm are given in the next section.

VI. Experimental Results

Experimental simulations were performed to compare the performance of the soft competition scheme with that of the generalized Lloyd algorithm. The sources tested were digitized speech waveforms sampled at 8 kHz and first-order Gauss–Markov processes of the form $\mathbf{x}_n = \alpha \mathbf{x}_{n-1} + \mathbf{w}_n$, where $\mathbf{w}_n$ is an i.i.d. Gaussian process with zero mean and unit variance, and the coefficient α is the first autoregressive coefficient of the process $\mathbf{x}_n$. The values used for α were 0.9, 0.5, and 0 (the last is a Gaussian i.i.d. process).

Each algorithm was simulated on a Sun 3/260 computer with a floating point accelerator. One sweep of the SCS has approximately the same computational complexity as an iteration of the GLA. However, the time required for the SCS to converge was, depending on the temperature schedule, 3–10 times greater than the GLA. In an effort to make a fairer comparison, approximately equal computation time was allotted to each algorithm. Therefore, repeated runs of the GLA using different initial conditions were used, and the highest achieved SNR of these repetitions was used for comparison with a single run of the soft competition.

The time axis was segmented into frames whose lengths were measured by the number of sweeps they contained. Within each frame, the step-size schedule $a_i(n) = 1/n_i$ was used, where at the beginning of each frame all the counters n_i were reset to unity. The frames were chosen to increase in size by two sweeps after each update. This corresponds to resetting the codevectors' counters every time the current sweep index m equals a perfect square. After most resets of the step size, a temporary drop in the SNR was recorded, which was immediately (generally after one sweep) followed by a large recovery to an even higher SNR value (see, for example, Fig. 3). This behavior occurred because resetting the step sizes causes a relatively large change in the codevector locations, bringing them out of their near locally optimal positions as centroids, and hence temporarily reducing the SNR. The convergence to the different local minimum within the frame then recovers the lost SNR, and generally increases it. This reinitialization technique thus allows a more rapid convergence since the step size taken by the codevectors are larger, while the temporary loss of SNR is recovered generally within one sweep after the initialization. A plot of the step-size schedule as a function of the sweep number m is shown in Fig. 2 for a single codevector in a codebook of size four. Fig. 2 demonstrates the reinitialization

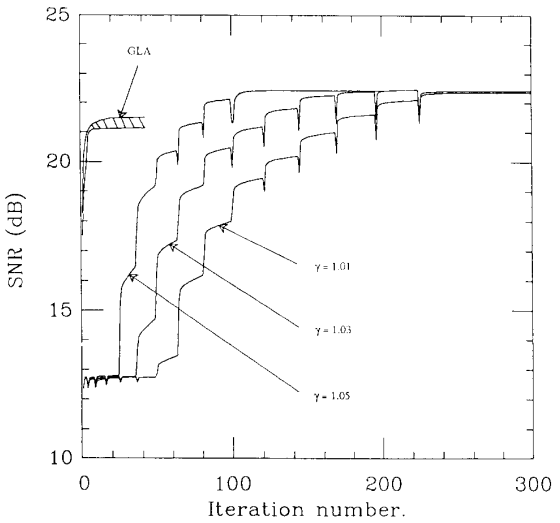


Fig. 2. Convergence of GLA and soft competition for various temperature schedules of the form $T(m) = T_0\gamma^{-m}n$, with $k = 2$, and rate = 3.5 b/sample. First-order Gauss-Markov source: $\mathbf{x}_n = 0.9\mathbf{x}_{n-1} + \mathbf{w}_n$. The best and worst of 10 initial conditions are plotted for the GLA, with the range shaded.

of the step size every time m equals a perfect square, and its monotonic decay within each frame.

The temperature schedule for the soft competition was $T(m) = T_0\hat{\gamma}^{-m/K}$, where $\hat{\gamma} > 1$, m is the sweep index, and K is the codebook size. That is, the temperature was decreased at the beginning of each sweep, and was kept constant within the sweep. The purpose of the codebook size K in the exponent of $\hat{\gamma}$ is to compensate for the fact that, at a given temperature, the probability $P_n(i)$ decreases as the codebook size increases. Without the factor K, the temperature cools too quickly for large codebooks, restricting the movement of the codevectors and "freezing" the system early. The performance of the system was not heavily dependent upon the choice of $\gamma = \hat{\gamma}^{1/K}$: $\gamma = 1.05$ worked consistently well for all the sources.

The initial temperature was chosen to be of the form $T_o = CK\sigma_x^2$, where σ_x^2 is the variance of the scalar source $\mathbf{x}$, and C is a constant to be determined experimentally. We found that choosing $0.1 < C \leq 0.4$ worked well in practice. It is important that the initial temperature not be chosen too large, for in such a case the codevectors may tend to merge together yielding a poor codebook. In fact, it can easily be shown that for $T_o = \infty$, the Euclidean distance between each pair of codevectors decreases as $1/n$, where n counts the number of training vectors within a sweep.

Fig. 3 shows the behavior of the SNR value versus the sweep number m for several different temperature schedules. Different values of γ caused differences in convergence time, but the final SNR was approximately the same for each of the temperature schedules. The range of GLA curves over 10 repeated trials with different initial conditions is shown in Fig. 3 by the shaded region, where the best performance curve of the GLA lies about 1 dB below

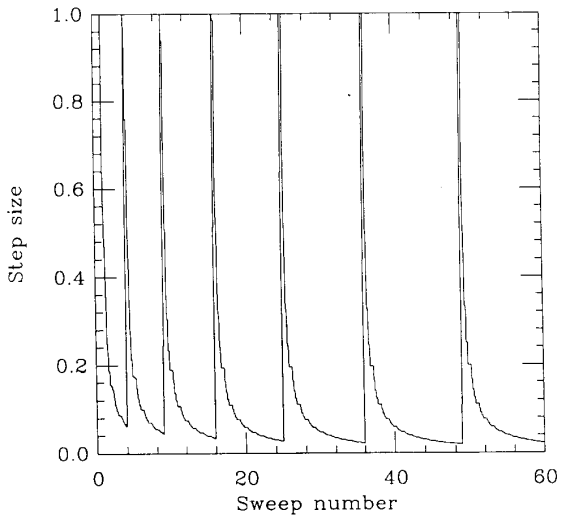


Fig. 3. Plot of step-size $a(n)$ for updating codevectors, versus sweep number of soft-competition algorithm for a first-order Gauss-Markov source, $x_n = 0.9x_{n-1} + w_n$, with $k = 2$, and rate = 1 b/sample.

TABLE I
PERFORMANCE COMPARISON BETWEEN GLA AND SOFT COMPETITION ALGORITHM FOR FIRST-ORDER GAUSS-MARKOV SOURCES. THE SOURCE DATA IS DERIVED FROM A RANDOM PROCESS OF THE FORM $x_i = \alpha x_{i-1} + w_i$, WHERE w_i IS INDEPENDENT WHITE GAUSSIAN NOISE

α	Vector Dim.	Codebook Size	GLA SNR (dB)	Soft Comp. SNR (dB)
0	8	256	5.8	6.0
	4	256	11.1	11.3
	2	64	15.3	15.7
	2	128	18.2	18.8
	2	256	20.7	22.0
0.5	8	256	6.9	7.1
	4	256	12.0	12.3
	2	64	16.0	16.4
	2	128	18.8	19.6
	2	256	21.6	22.8
0.9	8	256	11.9	12.1
	4	256	16.4	16.8
	2	64	18.9	19.6
	2	128	21.6	22.6
	2	256	24.9	26.0

the peaks of the soft competition curves. That is, when allowed the same amount of computation time, the soft competition provides a better codebook (i.e., a better local minimum) than the GLA. The same behavior of the SCS with respect to the GLA was consistently observed in all the experiments conducted. While Fig. 3 shows the SNR for 300 sweeps of the soft competition algorithm, that many sweeps are unnecessary in the design process. The peak SNR in the curve with $\gamma = 1.05$ occurs at about sweep number $m = 102$, at which point the algorithm could be terminated.

Vector quantizers were designed for vectors of dimensions 2, 4, and 8, with rates ranging between 1 and 4 b per sample. The results are shown in Table I for the Gauss-Markov sources and in Table II for the speech source. The SCS consistently outperformed the GLA over a wide range of sources and bit rates. As seen in Tables I and II, the gain in SNR of the soft competition over the GLA ranges between 0.2 to 2.4 dB. The gain in SNR was generally greater for higher bit rates and for sources with higher correlation.

TABLE II
PERFORMANCE COMPARISON BETWEEN GLA AND SOFT COMPETITION ALGORITHM FOR SPEECH SOURCE DATA

Vector Dim.	Codebook Size	GLA SNR (dB)	Soft Comp. SNR (dB)
8	256	5.8	6.0
4	256	11.1	11.4
2	64	18.2	19.7
2	128	21.4	23.1
2	256	24.0	26.4

One important aspect of the SCS is its relative insensitivity to the choice of an initial codebook. Figs. 4(a) and (b) show the evolution of codebooks with four codevectors in two dimensions, as the soft competition algorithm progresses. The source in each case is a Gaussian i.i.d. process whose training vectors are plotted with $\times$'s. In Fig. 4(a) the initial codebook is chosen at random from the training sequence, and in Fig. 4(b) the initial codebook is chosen to lie equally spaced on a vertical line. In both cases, the final codebooks are the same, though the trajectories taken to achieve the locally minimal state are quite different.

VII. CONCLUSIONS

We have demonstrated convergence properties of a basic form of the Kohonen learning algorithm for vector quantizer design and obtained conditions on the step-size schedules to guarantee that the centroid and nearest neighbor rules are satisfied. With this background, a new VQ design technique was introduced by modifying the neighborhood mechanism of the original KLA using stochastic relaxation principles. The new algorithm is an on-line method in which the codevector update is governed by a "soft" competition between the codevectors, which are updated simultaneously for each presentation of a training vector. Computer simulations demonstrated the effectiveness of the new soft competition scheme by comparisons with the conventional GLA for codebook design.

APPENDIX A

1. *Proof of Theorem 1*

We first consider the convergence of the sequence $A(0, n)$ to zero (condition (3.5a)).

Lemma 1: The sequence $A(0, n)$ converges to zero as $n \to \infty$ if and only if (3.6) is satisfied.

Proof: Since $a(n) \in [0, 1]$ for any n, it follows that also $A(0, n) \in [0, 1]$. Moreover, since $A(0, n) = [1, a(n)]A(0, n-1) \leq A(0, n-1)$, the sequence $A(0, n)$

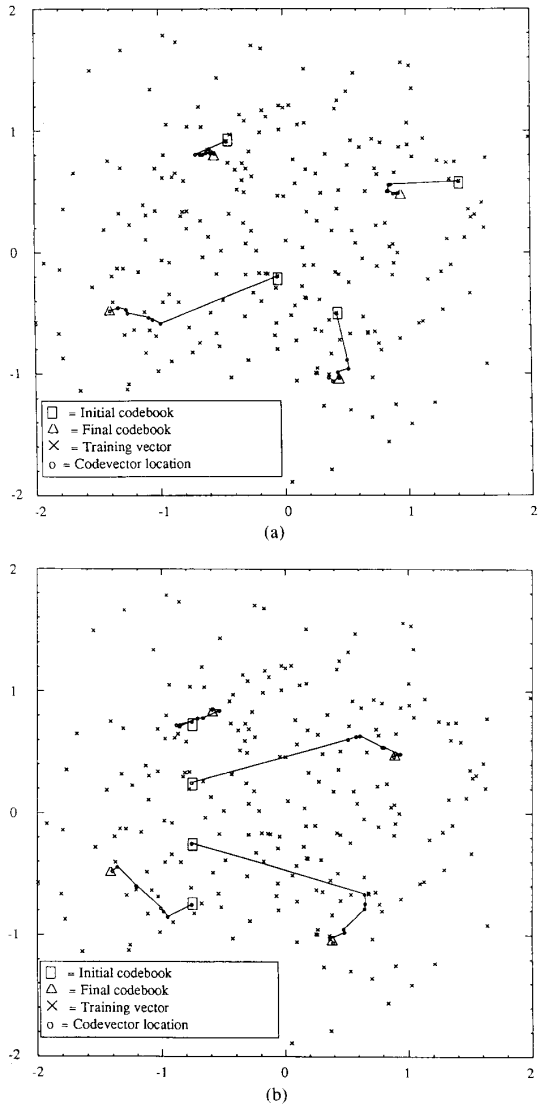


Fig. 4. (a) Plot of trajectories of 4 codevectors during the soft competition VQ design algorithm with random initial codeback. The locations of each separate codevector at the beginning of each sweep are connected by lines. The lengths of the connecting lines decreases with time as the temperature decreases. The source is a zero-mean, unit-variance, Gaussian i.i.d. process blocked into 2-dimensional vectors. (b) Plot of trajectories of 4 codevectors during the soft competition VQ design algorithm with initial codebook on vertical line in the plane.

decays monotonically, and hence converges. In order to give lower and upper bounds for $A(0, n)$, the following inequalities are used:

$$\ln (1 - x) < -x \qquad \text{for } 0 < x < 1 \tag{A.1}$$

$$\ln (1 - x) > -2x \qquad \text{for } 0 < x < 1/2. \tag{A.2}$$

Using (A.1), $A(0, n)$ can be upper bounded with

$$A(0, n) < \exp\left\{-\sum_{i=1}^{n} a(i)\right\} \tag{A.3}$$

from which it follows that $\lim_{n \to \infty} A(0, n) = 0$ if (3.6) is satisfied.

To show that (3.6) is also a necessary condition let us define i_o to be the least positive integer for which $a(i) < 1/2$, for all $i \geq i_o$, and let A_o be defined as

$$A_o = \sum_{i=1}^{i_o - 1} [1 - a(i)].$$

Then, using (A.2), $A(0, n)$ can be lower bounded with

$$A(0, n) > A_o \exp\left\{-2 \sum_{i=i_o}^{n} a(i)\right\} \tag{A.4}$$

so that, in the limit as $n \to \infty$, $A(0, n)$ will be bounded away from zero unless (3.6) is satisfied. □

We now examine the convergence of the sequence $r(n)$ as $n \to \infty$.

Lemma 2: $r(n)$ converges to unity as $n \to \infty$ if and only if (3.6) is satisfied.

Proof: From (3.2) it immediately follows that

$$q_N(n) = q_{N-1}(n)[1 - a(N)]. \tag{A.5}$$

Using (3.3b) it is straightforward to verify that the sequence $r(n)$ satisfies the difference equation

$$r(n) = [1 - a(n)]r(n - 1) + a(n); \qquad r(1) = a(1). \tag{A.6}$$

By defining the error sequence $e(n) \triangleq 1 - r(n)$, we get the recursion

$$e(n) = [1 - a(n)]e(n - 1); \qquad e(1) = 1 - a(1). \tag{A.7}$$

Since $a(n) \in [0, 1]$ and $e(n) \in [0, 1]$, it is evident that $e(n)$ is monotonically decreasing and bounded and therefore converges. Solving (A.7) with the given initial condition for $e(1)$ and using (2.7b) we get for all n

$$e(n) = A(0, n) \tag{A.8}$$

and the proof is completed by using Lemma 1. □

Combining the above two lemmas, it is concluded that (3.6) is a necessary and sufficient condition for the estimator to be asymptotically unbiased, as stated by Theorem 1. Furthermore, if $a(1) = 1$, then the estimator $y(N)$ is unbiased for any value of N since in this case, $e(N) = A(0, N) = 0$ for any N, and thus $r(N) = 1$ for any N.

2. *Proof of Theorem 2*

We first prove that $u(n)$ decreases monotonically and is bounded and thus converges (Lemma 3), and then proceed by showing that (3.6) is a necessary condition for the limit of $u(n)$ to be zero by upper bounding $u(n)$ $A(0, n)$.

Lemma 3: If $a(n)$ decreases monotonically then $u(n)$ decreases monotonically.

Proof: Following its definition in (3.4b), and using (A.5) it can be verified that $u(n)$ satisfies the difference

equation

$$u(n) = [1 - a(n)]^2 u(n-1) + a^2(n); \qquad u(1) = a^2(1). \tag{A.9}$$

Define the sequence

$$g(n) = \frac{a(n)}{2 - a(n)} \tag{A.10}$$

which decreases monotonically to zero with $g(n) \in [0, 1]$ for all $n \geq 1$. From (A.9) it immediately follows that

$$u(n) - u(n-1) = a(n)[2 - a(n)][g(n) - u(n-1)]. \tag{A.11}$$

Therefore, a necessary and sufficient condition for $u(n)$ to decrease monotonically is that $u(n-1) \geq g(n)$ for all $n \geq 2$. This condition is easily proved by induction. For $n = 2$, $u(1) = 1 \geq g(2)$, since $g(n) \leq 1$ for all n. Now, assuming that $u(n-1) \geq g(n)$ for some $n \geq 2$, we get (using (A.9) and (A.10))

$$u(n) \geq (1 - a(n))^2 g(n) + a^2(n) = g(n) \geq g(n+1) \tag{A.12}$$

which completes the induction argument. □

The sequence $u(n)$ decreases monotonically and is bounded and thus converges. To complete the proof of Theorem 2 we define a sequence $w(n)$ by the recursion

$$w(n) = [1 - a(n)]^2 w(n-1); \qquad w(1) = u(1). \tag{A.13}$$

It can be verified (by induction) that $w(n) \leq u(n)$ for all $n \geq 1$. Hence, by solving (A.13) for $w(n)$ in terms of $w(1)$ it follows that

$$u(n) \geq w(n) = A^2(1, n)u(1). \tag{A.14}$$

However, since the convergence of the sequence $A(1, n)$ is equivalent to that of $A(0, n)$, the proof is completed by appealing to Lemma 1. □

3. Proof of Theorem 3

The proof is obtained by assuming that (3.7) holds and showing that $u(n) \leq a(n)$, for any $n \geq 1$. This proves that $u(n)$ converges to zero since $a(n)$ does. First, note that (3.7) can equivalently be expressed for all $n \geq 2$ as

$$a(n-1) \leq \frac{a(n)}{1 - a(n)}. \tag{A.15}$$

We prove that $u(n)$ is upper bounded by $a(n)$ by induction. For $n = 1$, $u(1) = a^2(1) \leq a(1)$. Now, assuming that $u(n-1) \leq a(n-1)$ for some $n \geq 2$, we get (using (A.9) and (A.15)) that

$$u(n) - a(n) \leq [1 - a(n)]^2 \left[a(n-1) - \frac{a(n)}{1 - a(n)} \right] \leq 0 \tag{A.16}$$

which completes the induction argument. □

4. Proof of Theorem 4

By writing

$$u(N) - \frac{1}{N} = \sum_{n=1}^{N} \left[\left(q_N(n) - \frac{1}{N} \right) \left(q_N(n) + \frac{1}{N} \right) \right] \tag{A.17}$$

and partitioning the set of time instants $\{1, \cdots, N\}$ into the two subsets

$$N^+ = \left\{ n \in [1, N] | q_N(n) \geq \frac{1}{N} \right\}$$
$$N^- = \left\{ n \in [1, N] | q_N(n) < \frac{1}{N} \right\} \tag{A.18}$$

(A.17) can be upper bounded with

$$\begin{aligned} u(N) - \frac{1}{N} &\geq \sum_{n \in N^+} \frac{2}{N} \left(q_N(n) - \frac{1}{N} \right) \\ &\quad - \sum_{n \in N^-} \frac{2}{N} \left(\frac{1}{N} - q_N(n) \right) \\ &= \frac{2}{N} [r(N) - 1] = 0 \end{aligned} \tag{A.19}$$

in which equality is clearly obtained if and only if $q_N(n) = 1/N$ for all $n \in [1, N]$. □

5. Proof of Theorem 5

By Theorem 4 we can restrict attention to sequences $a(n)$ for which $q_N(n) = 1/N$ for all $1 \leq n \leq N$. Note that the equality $A(N, N) = 1$ in (2.7b) implies the boundary condition $a(N) = 1/N$. The requirement on $q_N(n)$ can be also written as ($1 \leq n \leq N$)

$$\frac{q_N(n)}{q_N(n-1)} = 1 \tag{A.20}$$

which, using (2.7b), can be written as the difference equation ($2 \leq n \leq N$)

$$a(n-1) = \frac{a(n)}{1 - a(n)}. \tag{A.21}$$

By induction it is verified that the unique solution for (A.21) with the specified boundary condition is given by $a(n) = 1/n$ for all $n \in [1, N]$. In light of (A.21), the sufficient condition (3.7) (or equivalently (A.15)) given in Theorem 3 is tight, and is achieved only if $a(n) = 1/n$. In this case $u(n)$ achieves its upper bound sequence $a(n)$ and we get $u(N) = a(N) = 1/N$, and the Cramer–Rao bound is achieved.

6. Proof of Theorem 6

We first show the equivalency between $r \leq 1$ and the sufficient condition of (3.7).

Lemma 4: If $a(n) = n^{-r}$ with $r > 0$, then (3.7) is satisfied if and only if $r \leq 1$.

Proof: Applying the given form for $a(n)$ to (3.7) and

rearranging yields the equivalent condition, for $n \geq 1$,

$$n^r + 1 \geq (n+1)^r. \tag{A.22}$$

Let $f(x) \triangleq x^r + 1 - (x+1)^r$ for $x \geq 1$, whose derivative is given by $f'(x) = r[x^{r-1} - (x+1)^{r-1}]$. Both $f(1)$ and $f'(x)$ are positive if $r < 1$, zero if $r = 1$, and negative if $r \geq 1$, from which it is concluded that for any $n \geq 1$, $f(n) \geq 0$ if and only if $r \leq 1$. □

The proof of Theorem 6 is then concluded by combining the above lemma with Theorems 1-3.

Appendix B

In this Appendix we provide a proof of (4.6).

For each i denote by s_m^i the state vector that would result from changing the value of the mth component to be $s_m = i$, while leaving the other components of s unaltered, and let $P(s_m^i)$ be the probability of attaining such a vector. $P(s_m^i)$ is obtained from (4.4) by substituting s_m^i for the state vectors. Note that $P(s_m^i)$ is also the probability of the joint event $\{s_m = i, s_n \; \forall \; n \neq m\}$. By applying Bayes' rule to (4.5), the conditional probability $P_m(i)$ can also be written, for $1 \leq i \leq K$ as

$$P_m(i) = \frac{P(s_m^i)}{\sum_{j=1}^{K} P(s_m^j)} = \frac{1}{\sum_{j=1}^{K} \frac{P(s_m^j)}{P(s_m^i)}}. \tag{B.1}$$

Applying the Gibbs distribution of (4.4), the above expression can be written as a function of the distortion measure as

$$P_m(i) = \frac{1}{\sum_{j=1}^{K} \exp\{-\beta[D(s_m^j) - D(s_m^i)]\}} \tag{B.2}$$

where $D(s_m^i)$ is the value of the distortion function $D(s)$ computed for the state vector s_m^i (for which $s_m = i$). The quantity $D(s_m^j) - D(s_m^i)$ is the change in the distortion function when a transition from state s_m^i ($s_m = i$) to state s_m^j ($s_m = j$) occurs. Since s represents a partition of the training set, the transition from s_m^i to s_m^j is obtained by reassigning x_m from R_i to R_j. We now compute the change in the distortion function resulting from such a transition.

Denote the partition implied by s_m^i as state A, and the partition of s_m^j as state B. Also, let R_i^A denote the ith partition region in state A, and y_i^A its centroid, and similarly for R_i^B and y_i^B. We are interested in the transition from state A to state B as described schematically in Fig. 5, where H_i is the set with cardinality N_i defined by

$$H_i = \{x_k \in T | s_k = i \quad \text{and } x_k \neq x_m\}$$

and similar definitions hold for H_j and N_j. The transition from A to B can be expressed by the relations

$$\begin{aligned} R_i^A &= H_i \cup \{x_m\} \\ R_j^A &= H_j \\ R_i^B &= H_i \\ R_j^B &= H_j \cup \{x_m\} \end{aligned} \tag{B.3}$$

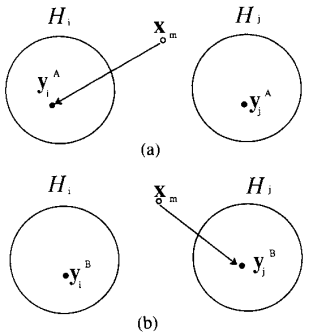


Fig. 5. Schematic description of the transition from state A to state B, in which the training vector x_m changes its affiliation from R_i to R_j.

for which the corresponding centroids are given by

$$\begin{aligned} y_i^A &= \frac{1}{N_i + 1}\left[x_m + \sum_{x \in H_i} x\right] \\ y_j^A &= \frac{1}{N_j} \sum_{x \in H_j} x \\ y_i^B &= \frac{1}{N_i} \sum_{x \in H_i} x \\ y_j^B &= \frac{1}{N_j + 1}\left[x_m + \sum_{x \in H_j} x\right]. \end{aligned} \tag{B.4}$$

The above expressions yield the relations

$$y_i^B = y_i^A - \frac{1}{N_i}(x_m - y_i^A) \tag{B.5a}$$

$$y_j^B = y_j^A + \frac{1}{N_j + 1}(x_m - y_j^A) \tag{B.5b}$$

and the change in the distortion function due to the transition from A to B is given by

$$\begin{aligned} D(B) - D(A) = {} & \|x_m - y_j^B\|^2 - \|x_m - y_i^A\|^2 \\ & + \sum_{x \in H_i} (\|x - y_i^B\|^2 - \|x - y_i^A\|^2) \\ & + \sum_{x \in H_j} (\|x - y_j^B\|^2 - \|x - y_j^A\|^2). \end{aligned} \tag{B.6}$$

By substituting the expressions for y_i^B and y_j^B from (B.5) into (B.6), the change in the distortion function can be written as

$$\begin{aligned} D(B) - D(A) = {} & \frac{N_j}{N_j + 1}\|x_m - y_j^A\|^2 \\ & - \frac{N_i + 1}{N_i}\|x_m - y_i^A\|^2. \end{aligned} \tag{B.7}$$

If the training set is large and we assume $N_i, N_j >> 1$, then substituting (B.7) into (B.2) yields the final expression, for $1 \leq i \leq K$

$$P_m(i) = \frac{e^{-\beta\|x_m - y_i\|^2}}{\sum_{k=1}^{K} e^{-\beta\|x_m - y_k\|^2}}. \quad \text{(B.8)}$$

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their extraordinarily thorough reviews and helpful comments.

REFERENCES

[1] A. Gersho, "On the structure of vector quantizers," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 157–166, Mar. 1982.
[2] R. M. Gray, "Vector quantization," *IEEE ASSP Mag.*, vol. 1. pp. 4–29, 1984.
[3] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84–95, Jan. 1980.
[4] S. P. Lloyd, "Least-squares quantization in PCM," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 129–137, Mar. 1982.
[5] R. M. Gray and E. D. Karnin, "Multiple local optima in vector quantizers," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 2, pp. 256–261, Mar. 1982.
[6] K. Zeger, J. Vaisey, and A. Gersho, "Globally optimal vector quantizer design by stochastic relaxation," *IEEE Trans. Signal Processing*, this issue, pp. 310–322.
[7] T. Kohonen, *Self-Organization and Associative Memory*. Berlin: Springer, 1984.
[8] N. M. Nasrabadi and Y. Feng, "Vector quantization of images based upon the Kohonen self-organization feature maps," in *Proc. 2nd ICNN Conf.*, vol. I, 1988, pp. 101–105.
[9] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 IRE WESCON Conv. Rec.*, part 4, 1960, pp. 96–104.
[10] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
[11] P.-C. Chang and R. M. Gray, "Gradient algorithms for designing predictive vector quantizers," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 679–690, Aug. 1986.
[12] A. Gersho, "Adaptive vector quantization," *Ann. Telecommun.*, vol. 41, no. 9–10, pp. 470–480, Sept.–Oct. 1986.
[13] E. Eweda and O. Macchi, "Convergence of an adaptive linear estimation algorithm," *IEEE Trans. Automat. Contr.*, vol. AC-29, pp. 119–127, Feb. 1984.
[14] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-6, pp. 721–741, Nov. 1984.
[15] K. Fukushima, "Cognitron: A self-organizing multilayer neural network," *Biol. Cybern.*, vol. 20, pp. 121–136, 175.
[16] S. Grossberg, "Adaptive pattern classification and universal recoding, Part I: Parallel development and coding of neural feature detectors," *Biol. Cybern.*, vol. 23, pp. 121–134, 1976.
[17] C. Von der Malsburg, "Self-organizing of orientation sensitive cells in the striate cortex," *Kibernetik*, vol. 14, pp. 85–100, 1973.
[18] D. E. Rumelhart and D. Zisper, "Feature discovery by competitive learning," *Cognitive Sci.*, vol. 9, pp. 75–112, 1985.
[19] D. DeSieno, "Adding a conscience to competitive learning," in *Proc. 2nd ICNN Conf.*, vol. I, 1988, pp. 117–124.
[20] H. Ritter and K. Schulten, "Kohonen's self-organization maps: Exploring their computational capabilities," in *Proc. 2nd ICNN Conf.*, vol. I, 1988, pp. 109–116.
[21] H. L. Van Trees, *Detection, Estimation, and Modulation Theory*. New York: Wiley, 1968.
[22] T. Kohonen, "The 'neural' phonetic typewriter," *Computer*, vol. 21, pp. 11–22, 1988.
[23] G. D. Tattersall, P. W. Linford, and R. Linggard, "Neural arrays for speech recognition," *Brit. Telecom. Technol. J.*, vol. 6, pp. 140–163, 1988.
[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA: M.I.T. Press/Bradford Books, 1986, ch. 8.
[25] H. Bourlard and C. J. Wellekens, "Multilayer perceptrons and automatic speech recognition," in *Proc. 1st ICNN Conf.*, vol. 4, 1987, pp. 407–415.
[26] H. Bourlard and C. J. Wellekens, "Speech dynamics and recurrent neural networks," in *Proc. IEEE ICASSP-89*, 1989.
[27] T. J. Sejnowski and C. R. Rosenberg, "NETtalk: A parallel network that learns to read aloud," Johns Hopkins Univ., Tech. Rep. JHU/EECS-86/01, 1986.
[28] A. Weibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 328–339, 1989.
[29] D. J. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1162–1168, 1988.
[30] L.-Y. Cun, "HLM: A multilayer learning network," in *Proc. 1986 Connectionist Models Summer School* (Carnegie-Mellon Univ., Pittsburgh, PA), 1986, pp. 169–177.
[31] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *J. Neural Networks*, vol. 1, pp. 75–89, 1988.
[32] A. H. Kramer and A. S. Vincentelli, "Efficient parallel learning algorithms for neural networks," in *Proc. NIPS Conf. 1988*, D. Touretzky, Ed., 1989, pp. 40–48.
[33] S. Kullback, *Information Theory and Statistics*. New York, Wiley, 1959.
[34] H. J. Sussmann, "On the convergence of a learning algorithm for Boltzmann machines," Rutgers Univ., Tech. Rep. SYCON-88-03, 1988.
[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 13, 1983.
[36] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, pp. 1087–1091, 1953.
[37] P. J. M. Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht, Holland: D. Reidel, 1987.
[38] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Carnegie-Melon, Tech. Rep., 1984.
[39] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: M.I.T. Press/Bradford Books, 1986, ch. 1.
[40] E. Yair and A. Gersho, "The Boltzmann perception network: A soft classifier," *J. Neural Networks*, vol. 3, no. 2, pp. 203–221, Mar. 1990.
[41] K. Zeger, "Corrections to 'Gradient algorithms for designing predictive vector quantizers,'" *IEEE Trans. Signal Processing*, vol. 39, no. 3, pp. 764–765, Mar. 1991.

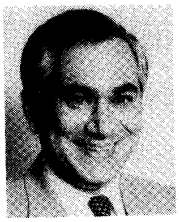
Eyal Yair was born in Haifa, Israel, on February 7, 1956. He received the B.Sc. and Ph.D. degrees in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 1982 and 1987, respectively.

From 1981 to 1983 he was a Teaching Assistant in the Department of Electrical Engineering at the Technion-Israel Institute of Technology. In 1983 he joined the IBM Israel Science and Technology Scientific Center. During the period 1987–1989 he was on leave as a postdoctoral Researcher at the Center for Information Processing Research, University of California, Santa Barbara.

Kenneth Zeger was born in Boston, MA, on August 18, 1963. He received both the S.B. and S.M. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (M.I.T.) in 1984, and both the M.A. degree in pure mathematics and the Ph.D. degree in electrical engineering from the University of California, Santa Barbara (UCSB), in 1989 and 1990, respectively.

During 1980–1981 he worked on adaptive antenna arrays for Z-A Inc. in Glenside, PA. He has worked on real-time speech recognition for Hewlett-Packard Company at the General Systems Division in Sunnyvale, CA, and on speech compression techniques at HP Laboratories in Palo Alto, CA, during the years 1982–1985. In 1984 he served as a consultant to Automatic Data Processing Company on digital network design. In July 1990 he joined the electrical engineering faculty at the University of Hawaii as an Assistant Professor. His present research interests include combined source/channel coding, speech and image compression, and computational complexity theory.

Dr. Zeger was awarded a four-year Faculty Development Graduate Fellowship by the American Electronics Association in 1985, was Co-Chairman of the 1990 IEEE Workshop on Communication Theory, held in Ojai, CA, and received an NSF Presidential Young Investigator Award in 1991.



Allen Gersho (S'58–M'64–SM'78–F'82) received the B.S. degree from the Massachusetts Institute of Technology in 1960 and the Ph.D. degree from Cornell University in 1963.

He is Professor of electrical and computer engineering at the University of California, Santa Barbara (UCSB), and Director of the Center for Information Processing Research at UCSB. He was at Bell Laboratories from 1963 to 1980. His current research activities are in the compression of speech, audio, images, and video signals. He holds patents on speech coding, quantization, adaptive equalization, digital filtering, and modulation and coding for voiceband data modems.

Dr. Gersho served as a member of the Board of Governors of the IEEE Communications Society from 1982 to 1985 and is a member of the Communication Theory Technical Committee and the Signal Processing and Communications Electronics Technical Committee of the IEEE Communications Society. He served as Editor of the IEEE COMMUNICATIONS MAGAZINE and Associate Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS. In 1980, he was awarded the Guillemin–Cauer Prize Paper Award from the Circuits and Systems Society. In 1983, he received the Donald McClennan Meritorious Service Award from the IEEE Communications Society, and in 1984 he was awarded an IEEE Centennial Medal. In 1987 and 1988 he received NASA Tech Brief Awards for technical innovation.